VŠB - TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2013 Ondřej Maceček

VŠB - TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA INFORMATIKY

Vyhledávání v obrázcích na základě barev

Image Search Based on Colors

2013 Ondřej Maceček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ondřej Maceček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vyhledávání v obrázcích na základě barev
Image Search Based on Colors

Zásady pro vypracování:

Cílem práce je provedení průzkumu existujících přístupů, návrh a implementace vybrané nebo vlastní metody a aplikačního prostředí pro experimenty.

1. Průzkum a popis existujících přístupů.
2. Návrh a implementace vybrané nebo vlastní metody.
3. Návrh a implementace počítačové aplikace pro provádění experimentů.
4. Návrh, realizace a hodnocení experimentů.

Seznam doporučené odborné literatury:

[1] Hasitha Bimsara Ariyaratne, Koichi Harada: Content based Image Retrieval using Spatial Relationships between Dominant Colours of Image Segments. VISAPP 2010:184-189.
Další dle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Miloš Kudělka, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Podpis: Macůrek Ondřej

Poděkování

Rád bych na tomto místě poděkoval všem účastníkům průzkumu, kteří mi byli nápomocni při získávání výsledků pro vývoj systému a dále bych rád poděkovat vedoucímu bakalářské práce Mgr. Miloši Kudělkovi, Ph.D. za jeho rady a čas, který mi věnoval při řešení dané problematiky.

Abstrakt

Cílem této bakalářské práce je vytvoření vhodného vyhledávacího systému založeném na detekci barev v obrázku.

První část je věnována již existujícím barevným modelům, jež můžeme použít při hledání barev.

V teoretické části práce jsou popisovány postupy tvorby vlastní palety barev a následné hledání jejich odstínů. Další podstatnou částí je rozbor problematiky segmentace obrázku pro zlepšení výsledků detekování barev.

Podstatnou součástí textu je také vlastní implementace, kde jsou popisovány veškeré postupy, jak teoretickou formou, tak praktickou. Setkáme se zde například s vývojem barev a jeho následným zařazením do palety i experimentálním průzkumem mezi lidmi, založeném na vnímání odstínů barev z palety. Dále popis postupu segmentace obrázku a jeho následné logické sloučení tak, aby výsledná detekce barev byla co nejúčinnější.

Klíčová slova: Detekce barev, Barevný model, Segmentace obrázků, Voxel, Vzdálenost vektorů

Abstract

The purpose of this bachelor´s thesis is creation of suitable searching system based on colours detection in the image.

The first part is dedicated to already existing colourful models which can be used to search for colours.

In the theoretical part of thesis are described methods of own color palette creation and then searching for tone of each colour. The next very important part are analysis of image segmentation for result improvement in colour detection.

An important part of the work is also own implementation where are described all procedures in theoretical and practial form. We will see the colour development followed by classification into the palette, an experimental research of people based on perception of colour tones from the palette. Then there is description of image segmentation´s process and its logical merging to get as effective results of colour detection as possible.

key words: Color detection, Color model, Image segmentation, Voxel, Distance vectors

Obsah

1 Úvod

21. století se vyznačuje velkým rozvojem automatizace. Počítače začínají řídit téměř vše a tím se zvětšuje množství dat, které nás přímo zaplavují. Tyto data je nutné uchovávat na různých úložištích, odkud musí být k dispozici pro jejich následné použití. Se vzrůstajícím počtem množství dat, nám stoupá náročnost jejich vyhledávání. Proto jsem se v mé bakalářské práci zaměřil právě na problematiku vyhledávání dat, konkrétně obrázků.

Výsledkem mé práce je vytvoření vyhledávacího systému, pracujícího na principu detekce barev. Díky tomuto systému bude možné vyhledávat obrázky na základě barev, které se v nich vyskytují. Nebude již nutné obrázky třídit podle různých názvů. Všechny budou moci být v jedné databázi, odkud budou vyhledávány podle jednoduchého klíče - barvy. Vznikne také velká úspora času, který byl dříve věnován neustálému třídění. Způsob vyhledávání systému ocení zejména fotografové, designeři, novináři i tvůrci webových prezentací.

V prvních dvou částech textu jsou nastíněny již existující postupy používané pro tvorby programů na detekci barev. Seznámíme se se základními barevnými modely typu RGB, CMY, HSL, Lab a různými matematickými postupy, umožňující přestupování mezi jednotlivými modely. Patřičnou část zabírá také popis tvorby barevné palety, zastupující hlavní roli ve vyhledávání obrázků. S barevnou paletou se pojí také metody, které řeší přiřazování odstínů barev z palety jako např. přiřazování k nejbližší barvě pomocí Euklidovské vzdálenosti nebo pomocí Voxelů. Daná kapitola je věnována i problematice spojené s detekcí barev, čili segmentace obrázku, díky níž byly vylepšeny výsledky detekce obrázku. Jedná se o segmentaci zlatým řezem, detekci hran a segmentací založenou na hledání oblastí. V části vlastní implementace poznáme jednotlivé postupy při tvorbě barev, odstínů, segmentace a uložení do databáze. Zde se objevují dva experimenty spjaté s tvorbou barevné palety. Rozsáhlejší část zabírá popis hlavního testu, založeném na vnímání barev a jejich odstínů u lidí. Součástí jsou také obrázky, popis programu a výsledky. Dále byla užita metoda přiřazování barev z obrázku do palety, důležitá pro hledání barvy ve Voxelech. Právě Voxelové prostory byly výsledkem hlavního experimentu. V této fázi se vyskytl problém s rychlostí zpracovávání obrázku, jehož řešení je podrobně popsáno postupem a metodami, nezbytnými pro celkové zrychlení.

S celkovou detekcí barev je i spojena segmentace obrázku, v konkrétním případě popisuji segmentaci založenou na principu zlatého řezu. Po segmentaci je následné popsané spojení oblasti nalezených barev v jednotlivých oblastech do jednoho celku tak aby bylo možné vyhledávat tyto obrázky v uložišti.

Poslední kapitola je věnována popisu vlastní implementaci programu která je rozdělená na dvě části, a to na detekce barev který obsahuje popis jednotlivých tříd a konkrétní třídní diagram a vyhledávaní obrázku s popisem jeho tříd a náhledem třídního diagramu taktéž.

2 Barevné modely

Barevné modely určují, ze kterých základních barev se budou ostatní barvy skládat, jaký bude poměr jednotlivých základních barev a jakým způsobem se budou základní barvy míchat. Veškeré informace o barevných modelech jsem čerpal z následujících zdrojů [1, 3, 4, 11]

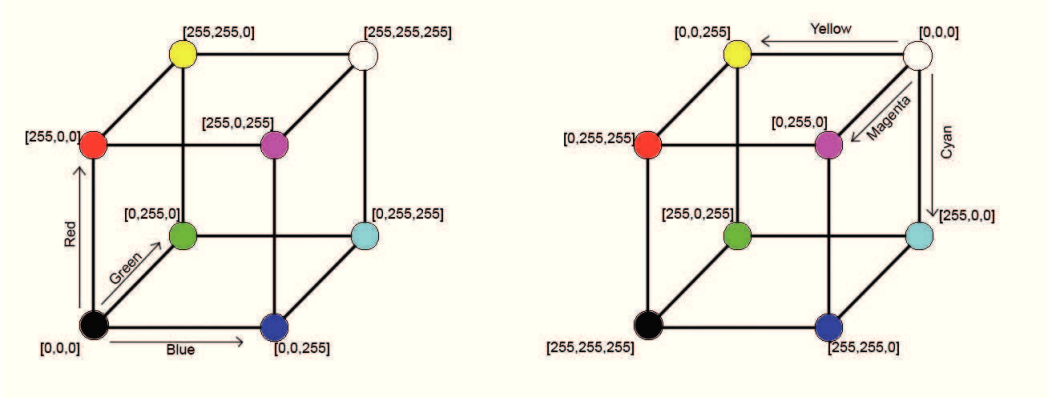
Lidské oko vnímá barvu působením na tři základní barevné receptory na sítnici a u většiny lidí jsou tyto receptory nejvíce citlivé na tři základní barvy, respektive jim odpovídající vlnové délky. Jedná se o : Červená (Red), Zelená (Green) a Modrá (Blue). Tomuto se nejvíce podobá barevný výstup na klasických *RGB* monitorech, kde jsou barevné odstíny vytvářeny kombinací právě těchto tří základních barev.

2.1 RGB barevný model

RGB model lze vyjádřit pomocí jednotkové krychle. V počátku *RGB* modelu leží barva černá, vyjádřena vektorem [0,0,0] a v protilehlém vrcholu barva bílá s vektorem [1,1,1]. Všechny barevné odstíny vznikají s mícháním třech základních barev: červená, zelená a modrá, jejichž intenzita je udána v intervalu $< 0, 1 >$. V počítačové grafice se tento interval dělí na 256 hodnot (0-255). Libovolná barva je udávána 24 bity (3x8). Barvy udávané pomocí 24 bitů nazýváme jako **true color**. V rámci **true color** můžeme získat 256^3 barev, přesně tedy 16 777 216 barev.

2.2 CMY barevný model

CMY model Obsahuje tři základní barvy - Tyrkysovou (Cyan), Fialovou (Magenta) a Žlutou (Yellow). Barevný rozsah tohoto modelu můžeme opět zobrazit jako jednotkovou krychli, která je do jisté míry shodná s krychlí *RGB*. Sčítání hodnot *CMY* ovšem odpovídá subtraktivnímu skládání barev, takže vrchol [1, 1, 1] reprezentuje černou barvu.



Obrázek 1: RGB barevný model, CMY barevný model

Převod mezi modely *RGB* a *CMY* je velmi jednoduchý. Vyjádříme-li bod v prostoru *RGB* jako tříprvkovou matici [r g b], pak bod v prostoru *CMY* určíme pomocí následujícího odčítání matic:

$$\begin{bmatrix} c \\ m \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} r \\ g \\ y \end{bmatrix}$$

2.3 HSV barevný model

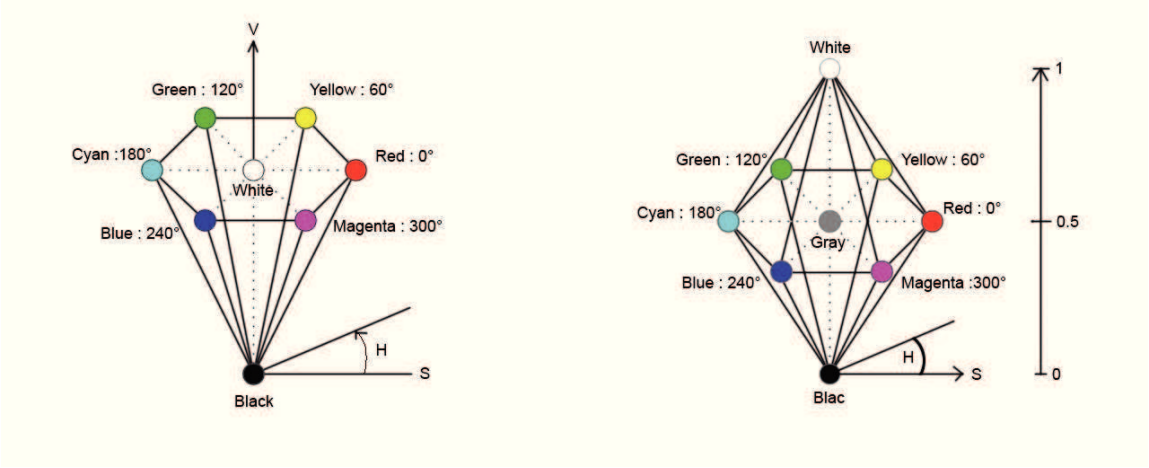
HSV barevný model, na rozdíl od *RGB* a *CMY*, lépe popisuje barvy, na které je člověk zvyklý. *HSV* má tři základní parametry: barevný tón, sytost a jas. Jinak řečeno, barevný tón určuje převládající barvu, sytost příměs jiných barev a jas příměs bíle barvy.

Geometrická reprezentace je pravidelný šestiboký jehlan. Jas barvy udává výška V, sytost barvy udává vzdálenost bodu od osy, jenž vede od středu podstavy do vrcholu jehlanu. Obal jehlanu má sytost rovnu jedné (tedy barvy se sytostí rovno jedné jsou nejvíc pestré). Barevný tón je úhel v rozsahu od 0 do 360 stupňů kde 0 nabývá barvy červené.

2.4 HSL barevný model

Nedostatky barevného modelu *HSV* odstraňuje barevný model *HSL*. Tyto dva modely jsou si velice podobné. Barva je definována třemi barevnými složkami stejně jako u *HSV*.

model je reprezentován dvoumi stejně velkými kužely se společnou podstavou. Tvar tohoto modelu odpovídá více reálným barvám. Má schopnost rozlišovat barevný odstín základní čisté barvy, který klesá (ztmavuje se), nebo stoupá (zesvětluje se) a to vše platí pro jednotlivé barvy. Zvyšování a snižování jasové složky spočívá s přidáváním světlého nebo tmavého pigmentu. Světlost udává výšku modelu (výška 0.5 odpovídá místu, kde jsou kužely spojeny).



Obrázek 2: HSV barevný model a HSL barevný model

Model *HSL* i *HSV*, na rozdíl od *RGB* a *CMY*, umožňují měnit pouze jeden parametr barvy, zatímco ostatní dva parametry zůstanou zachovány.

Pro převod z *RGB* do *HSL* musí být nejdříve znormalizovaný barevný vektor *RGB* do intervalu $< 0, 1 >$.

$$r = \frac{R}{255} \ , \ g = \frac{G}{255} \ , \ b = \frac{B}{255}$$

Dále je nutné je zjistit maximální a minimální hodnotu *RGB* složek. Následně se už provádí převod *RGB* do *HSL* následnými operacemi.

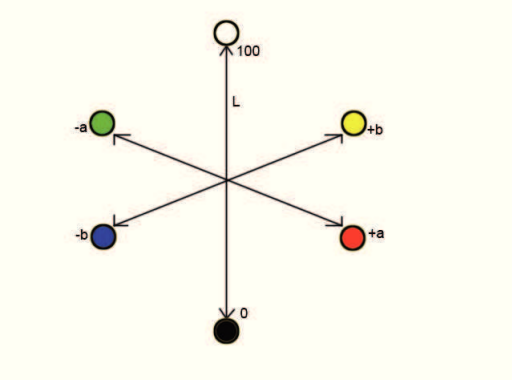
$$\mathrm{L} = \frac{1}{2} * (min + max)$$

$$S = \begin{cases} 0, & \text{Jestlize } min = max \\ \dfrac{max - min}{max + min} & \text{Jestliže } L > 0 \text{ a zároveň } L <= \dfrac{1}{2} \\ \dfrac{max - min}{2 - (max + min)} & \text{Jestliže } L > \dfrac{1}{2} \end{cases}$$

$$H = \begin{cases} 0, & \text{Jestlize } min = max \\ 60° * \dfrac{g - b}{max - min} + 0°, & \text{Jestliže } Max = r \text{ a } b \leq g \\ 60° * \dfrac{g - b}{max - min} + 360°, & \text{Jestliže } Max = r \text{ a } g < b \\ 60° * \dfrac{b - r}{max - min} + 120°, & \text{Jestliže } max = g \\ 60° * \dfrac{r - g}{max - min} + 240°, & \text{Jestliže } max = b \end{cases}$$

2.5 LAB barevný model

Model *Lab* popisuje barvu tak, jak ji normální člověk vnímá, nikoliv kolik a z kterých složek se barva skládá, Tento model je zcela nezávislý na zařízení, a právě proto se používá jako prostředek k transformaci z jednoho barevného modelu do jiného. Model *Lab* je třísložkový jako všechny ostatní. Jednotlivé složky mají význam: Světlost (Lightness, L) je v rozsahu 0 až 100 a popisuje jas bodu. 0 znamená černý bod, 100 znamená bílý bod, mezi kterými se nachází šedé odstíny. Složka *a* popisuje barvu bodu ve směru od zelené do modré (záporné hodnoty) a od červené do purpurové (kladné hodnoty). Složka *b* popisuje barvu bodu ve směru od modré do purpurové (záporné hodnoty) a od zelené do žluto-červené (kladné hodnoty).



Obrázek 3: Barevný model Lab

Převod z *RGB* modelu do *Lab* modelu se provádí ve dvou krocích. Zaprvé je nutné převést *RGB* model do modelu *XYZ*. Pro tento převod je nutné znormalizovat vektor *RGB* do intervalu $< 1, 0 >$ následujícími vztahy:

$$r = \frac{R}{255} \ , \ g = \frac{G}{255} \ , \ b = \frac{B}{255}$$

Pro co nejpřesnější převod z *RGB* do Lab je důležité normalizované složky *RGB* převést do modelu *sRGB*. Tento převod lze jednoduše zrealizovat následujícími podmínkami:

$$v = \begin{cases} \dfrac{V}{12.92} & V \leq 0.04045 \\ \left(\dfrac{V + 0.055}{1.055}\right)^{2.4} & V > 0.04045 \end{cases}$$

Hodnota bílého bodu v *sRGB* je označována **D65** a převod *sRGB* bílého bodu **D65** do *XYZ* lze provést následujícími vztahy:

$$\begin{aligned} X &= 0.4124564 \cdot r + 0.3575761 \cdot g + 0.1804375 \cdot b \\ Y &= 0.2126729 \cdot r + 0.7151522 \cdot g + 0.0721750 \cdot b \\ Z &= 0.0193339 \cdot r + 0.1191920 \cdot g + 0.9503041 \cdot b \end{aligned}$$

Posledním krokem je převod *XYZ* do *Lab* následujícím vztahem:

$$L = 116 \cdot f_y - 16 \quad , \quad a = 500 \cdot (f_x - f_y) \quad , \quad b = 200 \cdot (f_y - f_z)$$

kde pro jednotlivé f platí :

$$f_x = \begin{cases} \sqrt[3]{x_r} & x_r > \epsilon \\ \dfrac{k \cdot x_r + 16}{116} & x_r \leq \epsilon \end{cases}$$

$$f_y = \begin{cases} \sqrt[3]{y_r} & y_r > \epsilon \\ \dfrac{k \cdot y_r + 16}{116} & y_r \leq \epsilon \end{cases}$$

$$f_z = \begin{cases} \sqrt[3]{z_r} & z_r > \epsilon \\ \dfrac{k \cdot z_r + 16}{116} & z_r \leq \epsilon \end{cases}$$

Proměnné x_r, y_r, z_r vypočítáme následujícími vztahy:

$$x_r = \frac{X}{X_r} \quad , \quad y_r = \frac{Y}{Y_r} \quad , \quad z_r = \frac{Z}{Z_r}$$

Hodnoty ϵ a k jsou podle **CIE** standartu : $\epsilon = 0.008856 \quad , \quad k = 903.3$

X, Y, Z jsou trichromatické složky popisovaného barevného modelu *XYZ*, a Xn,Yn,Zn jsou trichromatické složky použitého normalizovaného světla, kde hodnoty normalizovaného světla **D65** jsou $Xn = 0.950455, Yn = 1, Zn = 1.088753$.

Každý z těchto barevných modelů nám může být nápomocen při tvorbě různých částí programu, například pomocí modelu *HSV* nebo ještě lépe *HSL* lze dobře promyslet barvy, které se nám budou vyskytovat v barevné paletě.

V dnešní době se nejčastěji vyskytují monitory, které zobrazují škálu *RGB* barev. Z tohoto důvodu je každý pixel z monitoru reprezentován *RGB* vektorem. V nadcházejícím textu se budeme zabývat rozebíráním problematiky čistě v *RGB* barevném spektru.

3 Průzkum a popis existujících přístupů

Pro docílení dobrých výsledků při detekci barev v obrázku a uchování pro jejich následné vyhledávání je nutné promyslet mnoho důležitých aspektů a postupů.

V první řadě je třeba definovat barvy, které budou v paletě a způsob, jak tuto paletu prezentovat. Dále je nutné zjistit, jak barvu z obrázku **true color** správně přiřadit do námi vytvořené palety a v neposlední řadě segmentovat obrázek tak, aby celý proces dosahoval co nejlepších a nejefektivnějších výsledků.

3.1 Metody vytváření barevné palety

Vytváření barevné palety je snad ta nejobtížnější operace z celého procesu. O tvorbě barevných palet jsem čerpal s následujícího zdroje[2].

Ze všech odstínů **true color**, musíme vybrat právě jenom ty, které budou reprezentovat vyhledávací systém. Vzhledem k tomu, že lidské oko dokáže rozlišit přibližně 10 milionů odstínů je téměř nemožné rozlišit všechny odstíny barev *RGB* modelu. Je tedy velmi nutné zvážit, které barvy si do palety barev zvolíme. Celé barevné rozhraní *RGB* lze popsat menší sadou barevných kategorií popsanou v *Tabulce 1.*

Color	R	G	B
Black	0	0	0
Sea green	0	182	0
Light green	0	255	170
Olive Green	36	73	0
Aqua	36	146	170
Bright Green	36	255	0
Blue	73	36	170
Green	73	146	0
Turquoise	73	219	170
Dark Red	109	36	0
Blue Gray	109	109	170
Lime	109	219	0
Lavender	146	0	170
Plum	146	109	0
Teal	146	182	170
Brown	182	0	0
Magenta	182	73	170
Yellow Green	182	182	0
Flouro Green	182	255	170
Red	219	73	0
Rose	219	146	170
Yellow	219	255	0
Pink	255	36	170
Orange	255	146	0
White	255	255	255

Tabulka 1: Výčet základních barev RGB modelu , zdroj [10]

Čím menší je počet barev, tím hruběji lze popsat rozpětí barvy, což může být výhodnější. Nicméně i ta nejrozsáhlejší rozpětí barev mohou ukázat, že i různé odchylky odstínu barvy jsou stále danou barvou.

3.2 Přiřazování true color do palety barev

Nyní, když už máme promyšlenou a vytvořenou paletu barev, je třeba pomocí některé metody určit, zda-li barva z obrázku patří do některé barvy z naší palety. V našem případě procházíme jednotlivé pixely námi předepsaným způsobem přes celý obrázek a pixely v **true color** vybíráme a následně je přiřazujeme do palety barev. I zde existuje několik různých metod, které se liší ve složitosti, paměťovou a časovou náročností a v neposlední řadě také kvalitou výsledného přiřazování barev. Ne vždy se dá dosáhnout výsledku aby každá barva **true color** byla přiřazena do palety.

Přiřazování **true color** z obrázku do palety může být provedeno následujícími metodami:

3.2.1 Vektorová vzdálenost

RGB barevný model má barvu reprezentovánou tří složkovým vektorem, kde každá složka zastupuje jednu základní barvu z modelu. Různým mícháním jednotlivých složek můžeme dostat všechny existující barvy. *RGB* model je tří složkový, proto ho můžeme považovat za prostorový vektor, pro jehož vzdálenost jednotlivých barev můžeme použít vypočetní metody vzdáleností vektoru v prostoru. Informace o vzdáleností vektorů jsem získal ze droje [7]

Jako první je vhodné uvést počítání **Euklidové vzdálenosti vektoru v prostoru:**

$$C_d = Min\sqrt{(X_r - Y_r)^2 + (X_g - Y_g)^2 + (X_b - Y_b)^2}$$

Je zřejmé, že ve vzorci pro výpočet vzdálenosti jsou použity dva vektory X, Y, kde vektor X je **true color** z obrázku a vektor Y je barva z naší vytvořené palety. Výsledkem by měla být minimální vzdálenost barvy z obrázku k barvě z palety. Pro výpočet vzdálenosti dvou vektorů lze použít i **Manhattonská vzdálenost** (někdy nazývána jako City Block), která je vyjádřena jako součet absolutních hodnot rozdílů jednotlivých složek vektorů:

$$C_d = Min(|X_r - Y_r| + |X_g - Y_g| + |X_b - Y_b|)$$

Zde se opět hledá nejmenší vzdálenost mezi dvěma barevnými vektory. Vektor X je barva nalezená v obrázku, vektor Y je barva z palety barev.

Výpočetní složitost vzdáleností vektoru je lineární, pro každý barevný pixel z obrázku musíme zjistit vzdálenost od každé barvy v naší paletě. Barvě s minimální vzdáleností bude připočten index jako hodnota četnosti v obrázku.

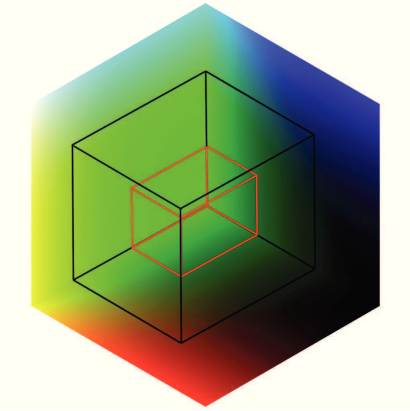
Tyto metody jsou velmi jednoduché na implementaci, ale bohužel pro potřeby detekce obrazu velmi nepřesné, a to z důvodu nelineárního zobrazování barev v *RGB* modelu a vnímání barev lidským zrakem. Proto byly navrženy specielní barevné modely, které umožňují měřit co nejpřesněji vzdálenost dvou barev. Jedná se například o model **Lab**. Kvalita přiřazování **true color** do palety barev přes model Lab ukazuje, že i přes nutnost použití neceločíselné aritmetiky, která značně zpomaluje výpočet vzdálenosti, se tato metoda převodu vyplatí.

3.2.2 Voxelové zpracování

Voxel *(Volumetric element, Objemový prvek)* je tzv. prostorový pixel umožňující uchovávat více odstínů jedné barvy v jednom elementu, tedy Voxelu. Představme si Voxel jako podmnožinu barev z celkové škály barev v *RGB* modelu. V praxi to znamená, že velikost Voxelu by měla být taková, aby všechny její vnitřní barevné odstíny byly v odstínu jedné základní barvy z palety. Zjistit velikost Voxelu ve všech osách barevného modelu je velice složitý proces, proto je třeba provést několik experimentů na vyhledávání a postupně měnit velikost os Voxelu.

Metoda Voxelového přiřazování je velmi rychlá. Má znatelně menší počet iterací oproti metodě vypočtu vzdálenosti barev. Pomocí Voxelu lze také docílit lepších výsledků díky úplné kontrole všech odstínů barev obsažené v něm. Nemůže dojít ke špatnému přiřazení barev vzhledem k nelineárnímu zobrazování barev v barevném modelu a vnímání barev lidským okem.

Jedinou nevýhodou Voxelového zpracování je nelineární zobrazení barev v *RGB* spektru. Voxel je jednoduchý šestistranný objekt (krychle, kvádr), a proto je prakticky nemožné uchovávat všechny odstíny jedné barvy v jednom Voxelu. Pokud by se však podařilo uchovat všechny odstíny barvy, začaly by se nacházet v určitých místech i odstíny jiných barev. Příkladem může být zelená barva, která je v prostoru *RGB* krychle velice častá a nelineárně rozšířená.



Obrázek 4: Maximální a minimální Voxel v *RGB* spektru zelené barvy

3.3 Segmentace obrázku

Segmentace obrázku je skupina metod, postavených na různých principech sloužících k rozdělení obrázku na oblasti se společnými vlastnostmi a které mají obvykle nějaký význam. Segmentace obrázku založená na detekci barev nám musí rozložit obrázek na logické celky určující skupinu barev v určitém místě. Potřebné informace o segmentaci obrázků jsem čerpal ze zdrojů [5, 6]

Příkladem mouhou být následující metody:

3.3.1 Nesegmentovaný obrázek

Jedná se o obrázek, který neprošel žádnou segmentací a oblastí obrázku je sám obrázek. Je tedy nutno procházet pixel po pixelu celým obrázkem a ukládat četnost barev. Výsledkem může být procentuální nebo poměrová četnost barev. Nesegmentovaný obrázek je tedy oblast, ve které nemůžeme určit o barvě žádné podrobné informace, jako např. kde se barva nachází, v jakém množství a jestli se vyskytuje v obrázku jen na jednom nebo více místech.

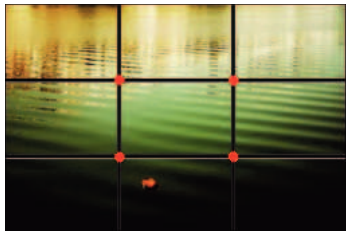
Je-li pro nás nedostačující vědět pouze četnost barev, musíme použít některé z následujících metod segmentace.

3.3.2 Zlatý řez obrázkem

Jinými slovy, poměr zlatého řezu je pojem týkající se kompozice obrázku. Jedná se o přímku rozdělující obrázek tak, aby na pohled působil co nejestetičtějším dojmem.

Místo, kde by měl přibližně zlatý řez ležet lze určit poměrně snadno. Stačí obrázek pomyslně rozdělit na třetiny, vodorovně i svisle. Průsečíky těchto třetin tvoří zlaté body obrázku.

Tímto rozdělením obrázku nám vzniknou 4 body v průsečících, ale také 9 stejně velkých oblastí, se kterými se dá samostatně pracovat. Tyto oblasti můžeme samostatně detekovat a určovat pravidla, pomocí kterých dosáhneme lepších výsledků a s každou oblastí můžeme pracovat jako s nesegmentovaným obrázkem, tedy uchovávat četnost barev. Na rozdíl od typického nesegmentovaného obrázku zde detekujeme 9 oblastí, díky kterým můžeme určit: jaká barva se kde vyskytuje, v kolika částech, v jakém množství a v jakém místě přibližně leží.



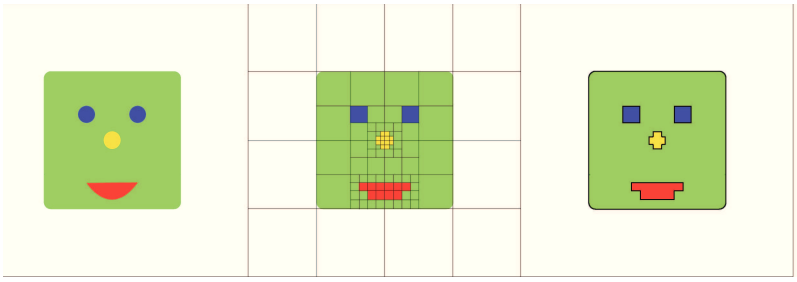
Obrázek 5: Zlaté body obrázku vytvořené pomocí zlatého řezu

Výpočetní složitost detekce oblastí zlatým řezem je stejná jako by sme pracovali s nesegmentovaným obrázkem. Narozdíl od nesegmentovaného obrázku zde můžeme dosáhnout o mnoho lepších výsledku.

3.3.3 Segmentace založená na hledání oblasti

Je to metoda založená na postupném dělení a následném spojování oblastí. Obraz je dělen do předem připravené struktury a následně se spojují se sousední oblastí pokud splňují dané kritéria. Nejčastěji používaná struktura je **quart tree**. Jedná se o stromovou strukturu, jejíž kořeny se větví do 4 dalších větví.

Zde je nutné určit kritérium, podle kterého bude obrázek postupně dělen. V případě detekce barev by jako kritérium mohla být změna jasu mezi sousedícími pixely. Při změně jasu se obrázek rozdělí do čtyř nových oblastí. Následně procházíme každou oblast zvlášť a kontrolujeme kriterium v každé nové oblasti. Tento proces trvá tak dlouho, dokud všechny oblasti nebudou splňovat námi dané kritérium. Až budou všechny vytvořené oblasti splňovat stanovené kritéria, začneme oblasti zpět spojovat.Tyto oblasti se můžou spojit za podmínky splňující stejné kritérium. Ve finálním výsledku získáme obrázek, jehož každá oblast splňuje dané kritéria.



Obrázek 6: Původní obrázek, rozdělení na oblasti a následné sloučení oblasti

Pomocí této segmentace je možné o každé barvě říct, ve které části obrázku se přesně nachází, v kolika částech a v jakém počtu se vyskytuje. Tyto podrobné informace nám mohou významě přispět k dobrým výsledkům. Problém však nástává ve složitosti výpočtu u obrázku, kde se barvy nachází v malém množství a na více místech. Může se tedy stát, že touto segmentací můžeme získat takový počet oblastí, který je roven počtu pixelů v obrázku. V tomto případě máme obrázek rozdělen na tolik segmentů, že jsme se v podstatě dostali do výchozí fáze. Navíc je celý proces velmi výpočetně náročný.

3.3.4 Detekce hran

Detekce hran je složitý a doposud nevyřešený problém v reálných scénách. Hranami rozumíme části obrázků, kde se prudce mění jasová hodnota.

Hranu můžeme chápat jako vlastnost obrazového bodu započteného jako funkci obrazu v okolí tohoto bodu. Hrana je pak reprezentovaná velikostí a směrem.

Celý proces detekce hran lze rozdělit do třech základních úkonů: filtrování, diferenciace a detekce. Šum, vzniklý při vzorkování obrazu, kvantování, rozmazání či nevhodném nastavení kamery může být částečně odstraněn vhodným filtrem. Diferenciace pak zvýrazní oblasti v obraze, kde je změna intenzity jasu obrazu významná. Nakonec jsou detekovány a lokalizovány body, kde je změna intenzity nejvýznamnější. Metody detekce hran lze rozdělit do dvou základních částí a to, metody jenž využívají první derivace a druhá která využívá druhou derivaci. Je-li použita první derivace, je výsledkem hranový gradient porovnán s prahem, který určuje, zda se jedná ho hranu či nikoliv. Použitím metody druhé derivace je výskyt hrany detekován a zda-li je prostorová změna v polaritě druhé derivace významná.

U metody detekce hran je možné docílit určování barvy všem objektů které se v obrázku vyskytují, jejich velikosti a pozici. Výsledky získané touto metodou budou velmi dobré. Složitost výpočtu a samotná implementace metody je mnohem složitější než u předchozích metod.

4 Vlastní implementace - Teorie zpracování

Má implementace programu prošla mnoha změnami počínajícími jak už od úplneho začátku nebo malými změnami v načítání struktur programu, pro zjištění lepšího způsobu přiřazování barevných odstínů do palety nebo změny základních barev v paletě barev.

4.1 Vytvoření palety obrázku

Je těžké popsat jak byla paleta vytvořena, protože původní verze palety byla nesčetněkrát měněna a vylepšována. Bohužel konečná verze stále není taková, aby vyhovovala všem. Každý člověk rozeznává barvy individuálně a má jiný cit na odstíny. Právě odstíny byly pro určité účastníky experimentu velkou překážkou a výsledky se někdy liší velmi znatelně.

V *tabulce 1* jsme již měli možnost vidět základní rozložení *RGB* barev, kde se většinou mění pouze jasová složka barvy. Dobrým příkladem je barva červená a tmavě červená. Obě tyto barvy jsou v základu červené, ovšem odstín barvy často dokázal zmást lidi natolik, že jeden z odstínů nebyl vyhodnocen jako červená ani tmavě červená, ovšem fyzicky o barvu červenou šlo.

Z tohoto důvodu jsem navrhnul velice jednoduchý program, který zobrazoval základních 25 barev z *tabulky 1*. Účastníci prvního průzkumu měli za úkol textově pojmenovat všechny barvy, které jim byli zobrazeny. Výsledky, které jsem získal přibližně od 10 účastníků byly takřka jednotné. Pomocí této metody se základní škála barev z *tabulky 1* snížila více než o polovinu, a to na 12. Výsledná paleta barev je podrobně popsána v *tabulce 2*.

Color	R	G	B
Červená	204	30	28
Černá	14	12	15
Modrá	43	95	200
Šedá	148	150	141
Zelená	73	183	81
Tyrkysová	61	210	211
Hnědá	148	86	34
Fialová	134	47	174
Žlutá	227	223	58
Růžová	231	90	171
Oranžová	235	129	28
Bílá	241	240	224

Tabulka 2: Barvy palety po prvním průzkumu

Zde se nabízí otázka, jak jsem určil *RGB* hodnoty? Tyto hodnoty byly na samém začátku vývoje programu zcela jiné. Výsledné barevné *RGB* vektory byly změněny až na úplném závěru vývoje, kdy už proběhl další průzkum na vnímání barevných odstínů těchto právě 12 barev.

Podrobným popisem druhého průzkumu se budeme zabývat v další části textu.

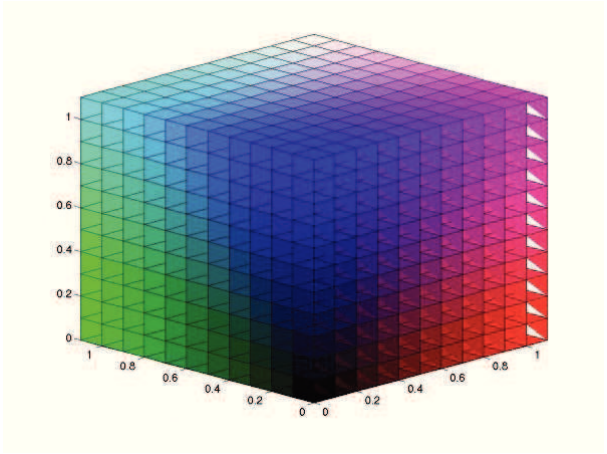
4.2 Detekce barev v obrázku - Voxelové zpracování

Pro přiřazování **true color** z obrázku do palety jsem použil Voxelovou metodu, která je velice rychlá a máme zde pod kontrolou všechny odstíny barev, které si sami nadefinujeme. Ovšem i tato metoda má pár nedostatků, proto jsem ji rozšířil o několik metod které napomáhají k lepšímu výsledku.

Jak již bylo zmíněno u Voxelového přiřazování barev, je tu velký problém s určením takového Voxelu, který by nám pojmul celou škálu odstínu určité barvy a přitom neobsahoval odstíny z jiné barvy.

Jak tedy docílit co nejlepších výsledků u definování velikosti Voxelu v *RGB* barevném modelu tak, aby obsahoval jen tu danou barvu, kterou potřebujeme? - Například tak, že pro jednu barvu nadefinujeme více Voxelů. Tímto způsobem se můžeme vyhnout omezujícímu šestistěnnému prostoru a skládat Voxely podle toho, jak je barva v *RGB* modelu rozložená.

Jak ale určit Voxely pro barvu z palety? - Tento problém jsem vyřešil druhým experimentálním průzkumem mezi lidmi, protože každý člověk má jiné vnímání barev a jejich odstínů a z tohoto důvodu to sám programátor nemůže určit.



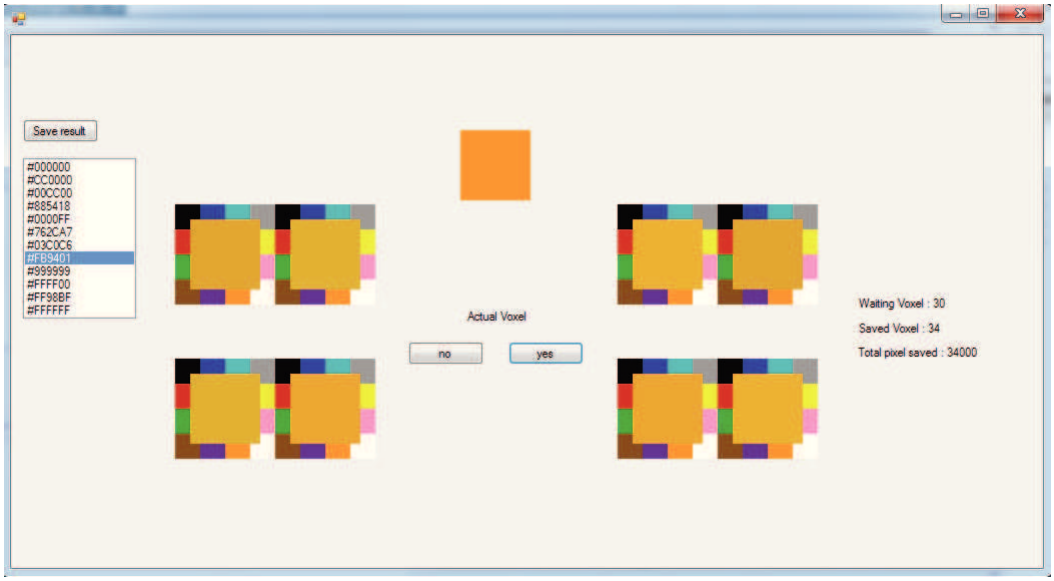
Obrázek 7: Ukázka možného rozložení Voxelu v RGB modelu

4.2.1 Experimentální průzkum tvorby Voxelu

K určení všech odstínů barev z palety, které budou reprezentovány Voxely, jsem sestrojil program, který mi byl velmi nápomocen. V základu programu jsem sestavil seznam barev *viz tabulka 2*, ale jejich *RGB* hodnoty *viz tabulka 1* tak, aby mezi jednotlivými barvami mohl účastník překlikávat a určovat dané odstíny základní barvy z palety.

Pro každou barvu program vypočítal základní Voxel o stejné velikosti všech hran. Velikost hrany bylo důležité pečlivě promyslet, aby výsledek experimentu vyšel v co nejlepších hodnotách. Bylo nutné nastavit velikost hrany co nejmenší. V ideálním případě by měla být hrana nastavena na velikost 1. Každý Voxel o velikosti 1 všech *RGB* os reprezentuje pouze jeden odstín z *RGB* modelu. V každém případě nemůžeme po účastnících chtít, aby nám rozhodli o každém odstínu barvy. Je tedy nutné určit takovou hodnotu, která by vyhovovala osobám vyplňujícím experiment a nám dala co nejpřesnější výsledky. Zvolil jsem tedy nakonec konstantní hodnotu hrany na 10. Tímto nám bude každý Voxel reprezentovat tisíc odstínů barevného *RGB* modelu.

Na začátku testu byl pro každou barvu z palety účastníkovy vygenerován základní Voxel o velikosti každé hrany 10, jehož středem je *RGB* vektor z *tabulky 1.* Každý z těchto Voxelů následně čekal na potvrzení účastníkem. Pokud ho potvrdil jako platný (odstíny Voxelu se shodují se základní barvou z palety) program vygeneroval nové Voxely, které měly vždy jednu společnou stranu s potvrzeným Voxelem, tedy maximálni počet vygenerovaných Voxelů je 6 (v případě, že se nenachází na okraji *RGB* krychle, nebo již takový Voxel nebyl dříve vytvořen). Tento proces pokračoval tak dlouho, dokud se nezačaly mezi Voxely objevovat odlišné odstíny a účastník tak nemohl odstíny potvrdit. Když už nebyly k potvrzení žádné nové odstíny, účastník jen uložil všechny hodnoty a mohl začít s další barvou z palety. Uložené výsledky by měly být v takovém formátu, aby s nimi bylo možno dále pracovat v následujícím vývoji programu na detekci barev v obrázku.



Obrázek 8: Náhled programů v experimentu pro průzkum odstínů barev

Pro zlepšení výsledků v experimentu byl v programu vytvořen náhled pro každou barvu z palety, ve kterém se fyzicky posuzovaná barva vyskytuje. Účastník tak mohl neustále vidět, kterou barvu to vlastně posuzuje a tím se kdykoliv ujistit. Velmi podobné odstíny totiž způsobovaly matení oka a mnohdy účastník vyhodnotil odstín špatně. Dále byly v programu vytvořeny náhledy osmi rohových odstínů posuzovaného Voxelu tak, aby účastník měl stále přehled, jak vypadají všechny krajové odstíny a mohl se tak správně rozhodnout. Každý z těchto osmi náhledů bylo obkresleno rámečkem, který obsahoval všech 12 základních barev. Tento krok byl nutný z důvodu přesnějších výsledků experimentu. Například, pokud vidíme šedou barvu na bílém podkladu zdá se nám více tmavá, než když je stejná šedá barva na černém pozadí.

Po provedení experimentů máme uloženy všechny odstíny barev podle toho, jak je účastníci vnímali. Bylo vhodné ještě s každou osobou překontrolovat jejich výsledky v programu na detekci barev a ukázat jim, jak jejich výsledky vnímání určují barvy v obrázku. Někdy bylo na místě ještě pár odstínů doladit, aby bylo vyhodnocení co nejpřesnější. V následující tabulce jsou výsledky experimentu. User 1 - 7 jsou účastníci a u každé barvy je počet nadefinovaných odstínových Voxelů.

Color	user 1	user 2	user 3	user 4	user 5	user 6	user 7	Součet
Červená	119	190	145	204	138	199	105	1 100
Černá	8	10	11	5	8	5	8	55
Modrá	1 260	1 109	1 305	998	1 321	1 153	1 099	8 245
Šedá	89	70	98	167	101	132	164	821
Zelená	2564	1780	1603	2328	1906	2602	1762	14 545
Tyrkysová	89	72	143	71	98	72	102	647
Hnědá	228	80	251	131	198	90	231	1 209
Fialová	1 071	1 209	1 021	1 240	980	1 303	1 156	7 980
Žlutá	135	97	140	138	135	89	179	913
Růžová	604	165	281	521	489	642	283	2 985
Oranžová	344	338	220	289	251	346	362	2 840
Bílá	51	3	9	4	3	6	5	81
Součet	6 562	5 123	5 227	6 096	5 628	6 639	5 456	-

Tabulka 3: Výsledky průzkumu tvoření odstínu barev z plalety

Ve sloupci součet je zobrazeno, kolik bylo celkem uživatelem vygenerovaných Voxelů jedné barvy. Tento součet zahrnuje i redundantní Voxely, kterých je však třeba se zbavit.

V řádku součet je zobrazeno, kolik odstínů všech barev účastníci celkem potvrdili. Když tento součet vynásobíme číslem tisíc, dostaneme celkový počet odstínů z *RGB* modelu, které můžeme detekovat Voxelovou metodou. V celkovém součtu se všichni uživatelé dostali na jednu třetinu všech odstínů *RGB* modelu. Pro dosažení dobrých výsledků nebylo takové množství ovšem dostačující, proto bylo nutné spojit výsledky do jednoho celku.

Výsledný počet Voxelů pro každou barvu experimentu, po odstranění duplicit a spojení výsledků všech učastníků, je následující:

Červená:	294	**Černá:**	19	**Modrá:**	1 501
Šedá:	238	**Zelená:**	3 046	**Tyrkysová:**	513
Hnědá:	312	**Fialová:**	1435	**Žlutá:**	378
Růžová:	1 420	**Oranžová:**	402	**Bílá:**	51

Ve výčtu jednotlivých odstínů si nyní můžeme spočítat celkový počet Voxelů, který je 9 609. V tomto případě jsme se dostali na úroveň bez mála deseti milionů odstínů. Tento počet je již dostačující víme-li, že lidské oko rozezná přibližně deset milionů odstínů.

Po provedeném průzkumu a dalších následných úpravách výsledků s ním spojených máme pro každou barvu vyhrazený prostor *RGB* modelu, kde se tato barva nachází. Tento prostor nám uchovávají Voxely, které jsme z průzkumu získali. Zde však nastává problém s rychlostí detekce barev v obrázku, kterou velmi významně ovlivňuje větší množství Voxelů pro každou barvu.

Nyní při přiřazování **true color** z obrázku do palety musíme porovnávat všechny barvy z palety a jejich Voxely, které barvy obsahují. To je bohužel výpočetně velmi náročné z důvodu počtu iterací, které se zvýšily o násobek počtu uchovávajících Voxelů. Následující dvě metody popisují, jakým způsobem lze snížit počet iterací při detekci barev v obrázku Voxelovou metodou:

4.2.2 Shlukování Voxelů

Pomocí této metody se má zredukovat počet Voxelů pro každou barvu na co nejmenší počet a zároveň se nesmí změnit prostor, který nám uchovává odstíny barev původních Voxelů v *RGB* modelu. Je tedy nutné sepsat algoritmus, který nám tuhle metodiku umožní.

Je několik možností, jak shlukovat Voxely. Můj algoritmus pracuje na základě hledání společných ploch Voxelů. Tato metoda lze využít v případě, že mají všechny Voxely stejnou velikost hran a jejich pozice se mění vždy jen o velikost hrany ve všech osách *RGB*. Po načtení všech Voxelů jedné barvy vytvořené v experimentu je setřídím podle jedné z *RGB* os. Následně je načten jeden Voxel z množiny a kontroluji, jestli existuje Voxel stejné barvy se stejnými souřadnicemi jedné z ploch na vybrané ose, podle které jsem Voxely setřídil. Pokud takový existuje, sloučím ho společně s původním dohromady a vznikne úplně nový, uchovávající stejný prostor v *RGB* modelu jako předchozí dva Voxely. Tento proces je proveden na každém Voxelu, který doposud nebyl sloučen nebo překontrolován. Jsou-li všechny Voxely jedné *RGB* osy sloučeny. Tuto operaci mohu aplikovat na druhou a následně i na třetí osu. Proces by se měl opakovat dokola tak dlouho pro každou barvu, dokud se počet Voxelů nedostane na co nejmenší množství. Když už mám nalezen minimální počet Voxelových prostorů ve všech barvách, můžu jejich výsledky přeuložit na místo, odkud se načítaly z původní množiny barevných Voxelů.

Po provedení shlukovací metody jsem dostal následující výsledky počtu Voxelů:

Červená:	65	**Černá:**	6	**Modrá:**	273
Šedá:	66	**Zelená:**	405	**Tyrkysová:**	142
Hnědá:	69	**Fialová:**	218	**Žlutá:**	81
Růžová:	310	**Oranžová:**	66	**Bílá:**	18

Celkový počet Voxelů je snížen na 1 719, ale prostor každé barvy z palety je uchováván stále stejný. Pomocí jiných metod by šlo dosáhnout ještě lepších výsledků, nicméně tato metoda je velmi jednoduchá a snížila počet iterací na jednu sedminu z původního počtu.

Při počtu 1 719 iterací pro každý pixel z obrázku je čas na zpracování jednoho obrázku stále poměrně dlouhý je tedy nutné zrychlit algoritmus pomocí podmínky pomocného maximálního Voxelu.

4.2.3 Maximální Voxel

Maximální Voxel lze určit tak, že pro každou barvu z palety načteme všechny odstínové Voxely a při procházení jednotlivých barevných odstínů hledáme minimální a maximální hodnoty Voxelu na osách barevného modelu. Po nalezení minima a maxima nám na každé ose vznikne oblast, ve které se nacházejí všechny Voxely jedné barvy z palety.

Tato oblast slouží k prvotní kontrole při přiřazování **true color** do palety. Po načtení pixelu z obrázku hledám barvu, do které tento pixel patří. První kontrola bude spočívat v tom, zda se barva nachází v našem maximálním Voxelu hledané barvy. Pokud ano, můžu začít procházet

všechny Voxely reprezentující odstíny právě hledané barvy. Pokud ne, barva se vyloučí. Tento proces provádím na všech ostatních barvách z palety tak dlouho, dokud není nalezena barva, do které pixel patří.

Zrychlení pomocí podmínky maximálního Voxelu je velice jednoduché na implementaci a časové zrychlení detekce obrázku je obrovské. Maximální počet iterací jednoho pixelu z obrázku se zmenší na takový počet, který udává barva prezentována největším počtem Voxelů (ve všech případech to je barva zelená). Maximální počet iterací u jednoho pixelu je v mém případě 405.

4.2.4 Ladění chyb v experimentu

Ačkoliv se na experimentu tvorby Voxelu podílelo více lidí, nebyly stále nalezeny všechny odstíny barev z palety. Všichni účastníci měli možnost k nahlédnutí do jejich výsledků a mohli sami vidět, jak jejich výsledky při testování obstály. Bohužel účastníci většinou nebyli moc spokojeni. Důvodem mohlo být například ovlivňování barvy pozadím a nebo špatně zvolena velikost hrany generovaných Voxelů.

Doplnění nedostatků může být provedeno hledáním barvy z **true color** do palety barev pomocí **Euklidovské vzdálenosti vektorů**. Tato metoda má ovšem taky své nedostatky, proto je třeba stanovit určité omezení této metody. Každá barva z palety má v *RGB* modelu různě rozprostřené své odstíny. Z tohoto důvodu je dobré určit pro každou barvu maximální vzdálenost vektoru, aby nedošlo k chybnému přiřazení barvy do palety.

Místo, kde se bude nacházet začátek vektoru v *RGB* modelu je pro každou barvu určeno pomocí ohniska všech Voxelů dané barvy. Ohnisko lze jednoduše vypočítat průměrnou hodnotou vektoru středu všech Voxelů jedné barvy. Výčet ohnisek barev z palety se nachází v *tabulce 2.*

Velikost vektoru pro každou barvu může být následující.

Barva	Velikost vektoru
Červená	70
Černá	20
Modrá	50
Šedá	40
Zelená	90
Tyrkysová	60
Hnědá	40
Fialová	70
Žlutá	50
Růžová	40
Oranžová	70
Bílá	20

Tabulka 4: Délky vektoru pro barvy z palety

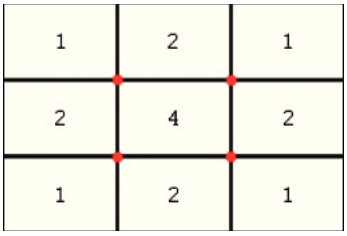
V tabulce lze vidět velikost vektoru každé barvy, která určuje jen barvy jejich odstínu. Pokud tedy není přiřazena barva z obrázku na základě voxelového zpracování je použita omezená metoda *Euklidovské vzdálenosti vektorů.* Výsledky již mezi lidmi odpovídají jejim vnímaní barev.

4.3 Segmentace obrázku

Přiřazování **true color** z obrázku do palety barev je vyřešen, nyní už jenom naimplementovat metodu, která nám řekne o obrázku více než jenom četnost barev, protože může nastat situace, kdy se bude jedna barva nacházet ve velmi malém množství na různých místech v obrázku, které lidské oko ani nezaregistruje. V tomto případě jistě nebudeme chtít, aby tato barvy byly detekovány jako barvy pro vyhledávání.

Pro segmentaci obrázku jsem použil metodu zlatým řezem. Tato metoda napomáhá určit, jakou část obrázku zrovna detekujeme, v kolika částech obrázku se nachází barva a v jakém množství se barva vyskytuje v oblasti.

Zlatým řezem dostaneme celkem devět oblastí v obrázku vzniklých průsečíky ve třetinách obrázku jak vertikálně tak i horizontálně. Vzniknou nám také čtyři body průsečíku, pomocí kterých jsem určil váhu oblasti. Váha oblasti jsem určil z důvodu prvotního zaměření lidského oka na obrázek. Barva uprostřed obrázku má rozhodně větší váhu pro vnímání než barva schovaná někde na okrajích nebo rozích obrázku. Váhu obrázku lze určit podle uvážení, v mém případě jsem určoval váhu na základě zlatých bodů v obrázku. Podle počtu zlatých bodů, které má oblast ve svých rozích je určena váha oblasti. Rozložení váhy obrázku lze vidět na *obrázku 9.*



Obrázek 9: Rozložení váhy jednotlivých oblastí zlatého řezu

Kdyby byl výskyt barvy přepočítan váhou naznačenou na obrázku, nebyl by rozdíl v oblastech tak radikální, proto jsem váhu každé oblasti vynásobil dvěmi. Jak lze vidět z *obrázku 9.* , pomocí této segmentace lze nezávisle na sobě detekovat každou oblast zvlášť, to lze využít k lepším výsledkům celkové detekce barev bez jakéhokoliv omezení na rychlostí zpracování obrázku.

4.4 Výsledný přepočet četnosti barev v obrázku

Celý popsaný proces vlastní implementace znázorňuje, jak řešit dané metody na detekci barev. Nyní je třeba ustanovit pravidla, podle kterých se nám bude prezentovat četnost barev v celém obrázku.

Můžeme si vymyslet nespočet různých metod, jak barvám určit výskytovou hodnotu v obrázku. V případě mé vlastní implementace jsem vyšel ze základu váhy oblasti. Procentuálním výskytem barvy v každé oblasti a počtem oblastí, ve které se barva nachází.

Pro zjednodušení jsem si určil pět hodnot, které mi reprezentují procentuální výskyt každé barvy v každé oblasti obrázku. Rozdělení procentuálních hodnot je následující:

- Hodnota 0: V oblasti se nachází méně než 10 % barvy.
- Hodnota 1: V oblasti se nachází více nebo rovno 10 % a méně než 30 % barvy.
- Hodnota 2: V oblasti se nachází více nebo rovno 30 % a méně než 70 % barvy.

- Hodnota 3: V oblasti se nachází více nebo rovno 70 % a méně než 90 % barvy.
- Hodnota 4: V oblasti se nachází více nebo rovno 90 % barvy.

Při výpočtu procentuální hodnoty barvy je důležité tyto hodnoty zhodnotit podle toho, kde se tyto barvy nachází v obrázku. Každá vypočítaná hodnota barvy je vynásobena váhou určenou dané oblasti. Pro zpestření výsledku je třeba ještě vydělit tuto novou hodnotu celkovým počtem oblastí v obrázku, v mém případě devíti.

$$V = \frac{w \cdot p}{9}$$

Jednoduchý vzoreček nám znázorňuje, jak spočítat pro každou barvu v každé oblasti hodnotu výskytu, tedy ve vzorci naznačenou veličinu V. w je hodnota váhy oblasti násobená hodnotou p, která nám reprezentuje procentuální hodnotu výskytu barvy v oblasti. Je-li pro každou barvu vypočítána hodnota procentuálního zastoupení v oblasti, je dále třeba zjistit, v kolika oblastech se barva vyskytuje.

Pro tento přepočet hodnoty V jsem zvolil přenásobení hodnoty číslem z intervalu $< 1, 2 >$, kde nám minimální hodnota 1 udává, že v obrázku se daná barva nenachází. Hodnota tedy zůstane stejná a maximální hodnota 2 nám říká, že se barva vyskytuje ve všech oblastech obrázku. V tomto případě se nám hodnota V zvýší o dvojnásobek.

$$V = V \cdot \left(1 + \frac{h}{9}\right)$$

Konečný přepočet hodnoty V záleží na tom, v kolika oblastech se tato barva nachází. To nám značí veličina h

Z celého tohoto procesu přepočítávání oblasti výskytu barev nám vznikne po zaokrouhlení hodnotový celočíselný výčet v intervalu $< 0, 24 >$

Pro rozšíření uživatelských možností jsem z důvodu výskytu barvy zvolil metodu rozdělení daného intervalu na pět částí. Uživatel si sám může určit, kolik bude chtít barvy v daném obrázku. Rozdělení intervalu je následující.

- interval $< 0, 1)$ v obrázku se nevyskytuje hledaná barva. $Hodnota : 0$
- interval $< 1, 6)$ je hledaná barva v nepatrném množství. $Hodnota : 1$
- interval $< 6, 16)$ obrázek obsahuje větší množství hledané barvy. $Hodnota : 2$
- interval $< 16, 22)$ ve většině obrazku je hledaná barva. $Hodnota : 3$
- interval $< 22, 24 >$ je skoro celý obrázek v jedné barvě. $Hodnota : 4$

Pro zjednodušení jednotlivých intervalů jsem si vytvořil výčet hodnot 0 až 4, kde každá hodnota určuje jeden z intervalu popsaných výše.

Tyto hodnoty už byly poté ukládány do uložiště společně s obrázkem. Jejich následné hledání v obrázku je již velice jednoduché i pro samotné uživatele.

4.5 Ukládání a hledání zpracovaných dat

Pro uchovávání dat četnosti barev jsem vytvořil jednoduchou databázi, obsahující pouze jednu tabulku. Proceduru, funkci a sekvenční počítadlo přiřazující ID hodnoty pro každý řádek tabulky. Každý řádek tabulky popisuje pouze jeden obrázek, četnost jednotlivých barev v obrázku, název a relativní cestu k obrázku.

Metoda přímého uložení obrázků do databáze je funkční, ovšem z hlediska použití nekorektní (časová náročnost). Jako mnohem efektivnější řešení se nabízí ukládání relativních cest k danému souboru a obrázek ukládat do souboru. Je nutné dávat si pozor na názvy souboru (Obrázku) z hlediska přeukládání dat. Řešením problému může byt spojení ID záznamu v databázi a názvu souboru. V tomto případě nám vždy vznikne jedinečný název a nemůže tímto dojít ke kolizi přeukládání souborů. Pro tento účel je třeba vytvořit proceduru, která nám dokáže spojit název obrázku s ID řádku, popřípadě další operace.

Databáze uchovává hodnotu pro každou z uložených barev, jenž byla nalezena při detekci obrázku, jeho název a cestu umístění do úložiště. Díky uloženým výsledkům dosažených detekcí je možné vyhledávat pomocí více barevných kódů zároveň. Aby však mohlo být toto zrealizováno je nutno vytvořit korektní příkaz pro vyhledávání v databázi.

V databázi jsou zaznamenávány celočíselné hodnoty pro každou barvu v intervalu $< 0, 4 >$. Pro zlepšení výsledků vyhledávání proto můžeme použít funkci GREATEST. Tato funkce nám sečte všechny hodnoty, podle kterých jsou kladeny podmínky pro vyhledávaní záznamy. Následně použijeme syntaktickou konstrukci ORDER BY jazyka *SQL* pro seřazení záznamů.

Výsledný příkaz SQL jazyka pro červenou a zelenou barvu s výskytovou hodnotou 2 a menší by mohl vypadat následovně:

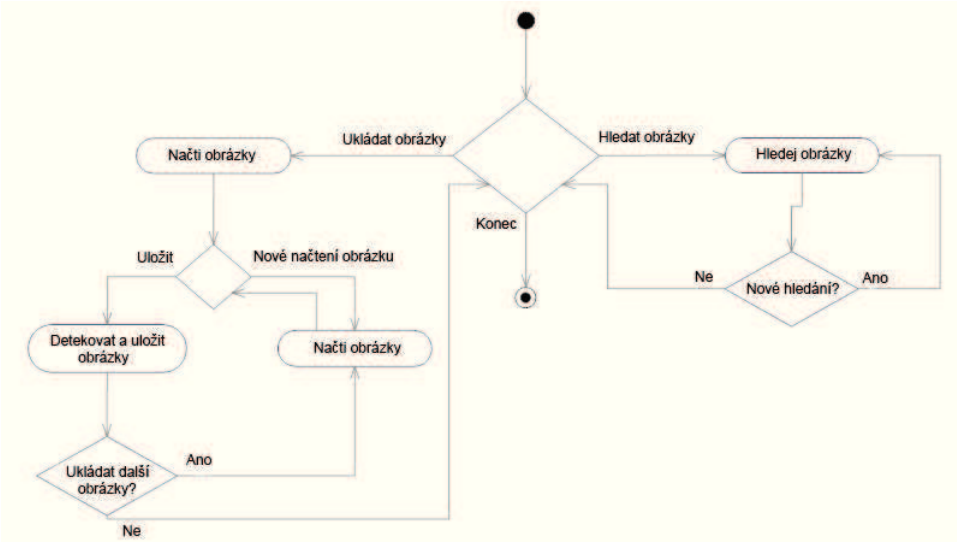
SELECT RelativníCesta, Greatest(ColorRed,ColorGreen) "greatest" FROM názevTabulky WHERE ColorRed <= 2 AND ColorGreen <= 2 order by greatest desc;

5 Vlastní implemenace - program

Vývoj této aplikace je vytvořeno v programovacím jazyce **C#** na platformě **Microsoft.NET Framework 4.5**. Využívám zde všechny potřebné komponenty pro vytvoření uživatelského rozhraní.

Celá implementace programu by se dala rozdělit na dvě hlavní části. Část, zabývající se detekcí barev v obrázku a část, vyhledávající již zdetekované obrázky. Obě tyto části mají společné pouze databázové uložiště a paletu barev.

- zpracování obrázku
- hledání zpracovaných obrázků



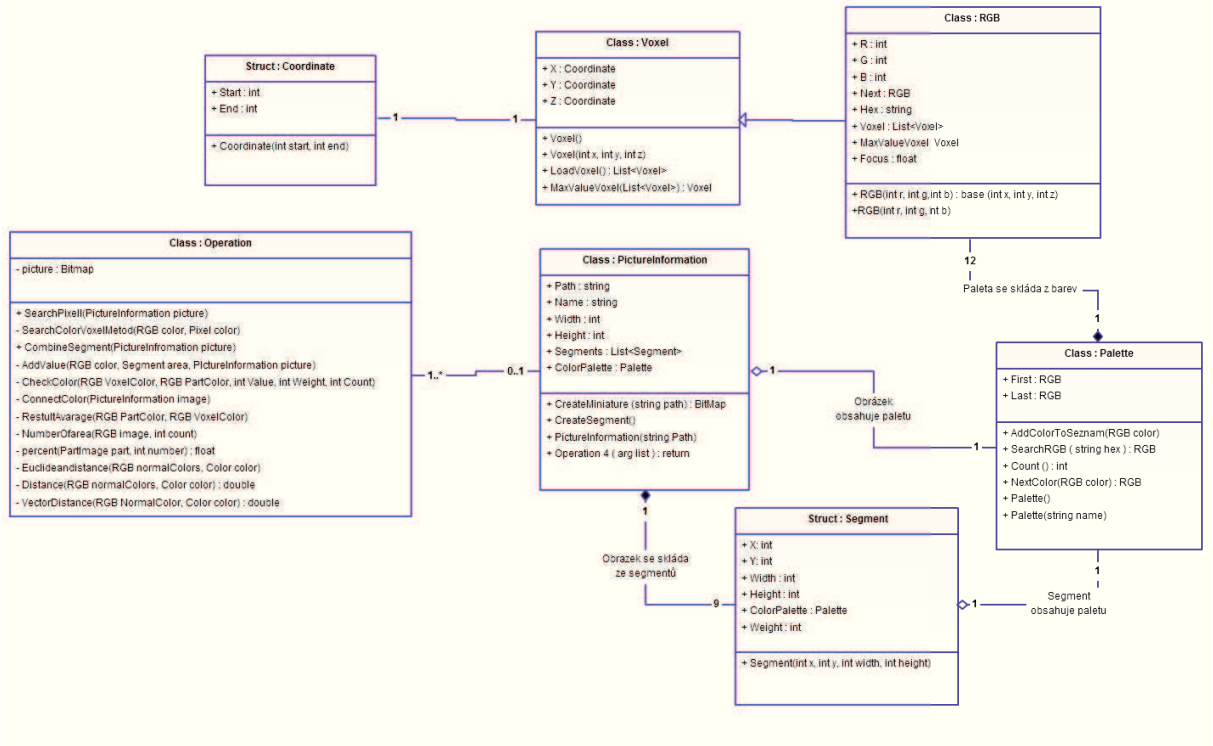
Obrázek 10: Diagram aktivit

5.1 Zpracování obrázku

Ke zpracování četnosti barev v obrázku je nutné vytvořit objekty, které budou uchovávat potřebné informace tak, aby proběhla úspěšně detekce barev v obrázku. Tyto objekty jsou rozděleny do následujících tříd:

Popis tříd

- Voxel
- RGB
- Palette
- PictureInformation
- Operation



Obrázek 11: Třídní diagram vzájemné závislosti tříd

Voxel: Pro reprezentaci Voxelu je vytvořena třída *Voxel* uchovávající nám velikost a souřadnice, kde se Voxel v *RGB* spektru nachází. Třídu *Voxel* tedy reprezentuje celkem šest souřadnic tak, že jednu z os *RGB* uchovávají právě dvě souřadnice. Jedna udává počátek a druhá konec Voxelu v jedné ose. Pojmenování šesti souřadnic je velice nepraktické a nepřehledné, proto jsem vytvořil pomocnou strukturu *Coordinate*, definující dvě souřadnice jedné z os Voxelu.

RGB: Třída *RGB* je navržena tak, aby reprezentovala základní informace o barvě, kterou chceme detekovat v obrázku. Základem této třídy je vektor, udávající souřadnice barvy v *RGB* spektru - výčet hodnot všech barev lze vidět v *tabulce 2*. Dále je třeba definovat rozsah odstínů pro jednotlivé barvy. Proto *RGB* třída dědí všechna data z výše zmíněné třídy *Voxel*. Všechny Voxely, vytvořené při druhém experimentu založeném na vnímání barev jsou postupně načítány ze souboru a uchovávány v objektu pole. Ihned po uložení všech Voxelů do pole je použita metoda, která toto pole postupně prochází a zjišťuje všechny minima a maxima souřadnic *RGB*. Tyto minima a maxima se na každé ose RGB uloží do nově vytvořeného Voxelu definujícího oblast *RGB* spektra, ve kterém se nacházejí Voxely uložené v poli. Dále pro *RGB* barvu definuji vektor se souřadnicemi ohniska, které jsou vypočítány z průměru vektoru středu všech Voxelů barvy a vzdálenost vektoru, jejíž hodnota je určená tak, aby od počátku ohniska byly nacházeny jenom čisté odstíny *RGB* barvy. Tyto vzdálenosti lze vidět v *tabulce 4*

Pallete: Pro reprezentaci všech barev vyhledávaných v obrázcích je navržena třída *Palette*, uchovávající v sobě všechny tyto barvy. V konstruktoru této třídy je pro každou barvu vytvořen nový objekt RGB s příslušnými parametry hodnot vektoru barvy. Všechny barevné objekty *RGB* jsou ukládány do jednosměrného lineárního seznamu. Pro jeden obrázek stačí vytvořit jednu

paletu barev, kde barvy reprezentují odstínové Voxely. V každém případě je třeba uchovávat hodnoty pro jednotlivé oblasti obrázku. Pro tento účel je vytvořen druhý konstruktor, vytvářející paletu barev již bez odstínových Voxelů.

PictureInformation: Třída *PictureInformation* uchovává potřebné informace o obrázku k jejímu načítání, procházení po jednotlivých oblastech a uchovávání barevné složky v oblasti i celém obrázku. K tomuto účelu třída uchovává hodnoty šířky a výšky obrázku, relativní cestu k načtení obrázku, paletu barev s Voxely a vhodnou metodu. Díky ní měníme velikost obrázku v poměru jejích stran pro vytváření miniatur obrázku a velikost určenou pro zpracování.

Pro segmentaci je nutné rozdělit obrázek na devět stejně velkých oblastí. K uchování informací o tom, kde se oblast nachází a jakou má velikost výšky a šířky je vytvořena struktura *Segment*, která tyto informace uchovává. K této struktuře je také vytvořena nová paleta barev, v níž už nemusí být žádná informace o Voxelech. Je vytvořena z důvodu uchovávání četnosti barev v jednotlivých oblastech.

Operation: Statická třída *Operation* je určena výhradně na detekci barev v obrázku. Zde není třeba uchovávat další data, aby mohla být provedena detekce barev v obrázku. Základní metoda přijímá parametr objektu *PictureInformation*, který je již zpracován a uchovává v sobě veškeré informace. Nejdříve se načte první oblast obrázku a cykly typu **FOR** procházejí oblastí pixel po pixelu. Při každém načteném pixelu se použije následující metoda, kontrolující postupně maximální Voxely všech barev z palety. Pokud platí podmínka, tedy pixel se nachází v maximálním Voxelu barvy, budou procházeny všechny Voxely nalezené barvy. Může nastat situace, že není nalezen Voxel barvy. V tomto případě je potřeba načíst další barvu. Pokud se stane, že barva nebude nalezena v žádném Voxelu naší palety, použijeme metodu, v níž se přepočítají všechny vzdálenosti vektoru barev z palety od barvy pixelu. Nejbližší vzdálené barvě z palety od pixelu je vytvořena další podmínka. Je-li vzdálenost menší než nadefinovaný vektor maximální vzdálenosti vektoru, bude se tento proces provádět pro každou oblast a pro každý pixel v oblasti. Je-li nalezená barva připadající barvě pixelu z obrázku, bude této barvě připočítán index četnosti v oblasti.

Celá struktura programu je naimplementována tak, že stačí vytvořit pouze jeden objekt *PictureInformation*. Tento objekt již načte všechny potřebné informace, aby mohla být provedena detekce barev v třídě *Operation*. V konstruktoru *PictureInformation* je vytvořený objekt *Palette*, tento objekt opět v konstruktoru vytváří celou škála barev (*RGB* objektu), které objekt *Palette* má obsahovat. Pokud je nutné uchovávat odstíny jednotlivých barev z palety, *RGB* třída načte v konstruktoru všechny Voxelové hodnoty.

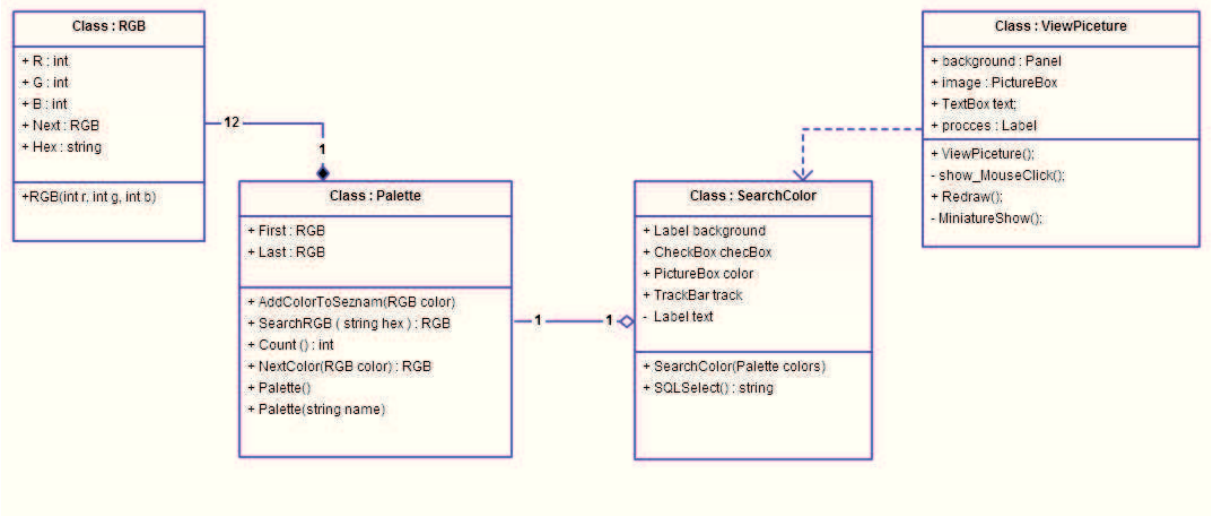
Kompletní vývojový diagram můžete vidět v *příloze A*.

5.2 Hledání zpracovaných obrázků

Aby mohli uživatelé vyhledávat obrázky, které již prošli detekcí barev, musí jím být zobrazena paleta barev, podle které jsme detekovali obrázky. Dále je potřeba vytvářet dynamický **select** podle požadavků uživatelů a na závěr musí být výsledek alespoň trošku uživatelsky dostupný, tedy obrázky se budou zobrazovat v určitém formátu, aby byl vidět výsledek. Pro tuto implementaci jsem navrhl čtyři třídy, zpracující veškeré informace od uživatele. Tyto informace se zašlou do databáze a zobrazí výsledek.

Popis tříd

- RGB
- Pallete
- ViewPicture
- SearchPicture



Obrázek 12: Rozložení váhy jednotlivých oblastí zlatého řezu

Třída RGB a Palette: Tyto třídy jsou již popsány výše. Zde u vyhledávání obrázku, stačí v Palette pro každou barvu vytvořit pouze barevny vektor, jehož barva bude vyhledávaná.

Třída SearchPicture: Tato třída vytváří uživatelský nástroj, s jehož pomocí definujeme informace potřebné k vyhledávání. Tento nástroj je navrhnut tak, aby uživatel měl co nejsnadnější práci pro výběr barvy, jenž chce vyhledávat. Velmi jednoduše lze také nastavit množstevní hodnotu výskytu barvy v obrázku.

V konstruktoru této třídy je vytvořen objekt *Palette*, v němž jsou uloženy všechny barvy z palety v lineárním seznamu. Tento seznam je celý vykreslen tak, aby mohl uživatel vybrat jednu nebo více z barev pomocí **CheckBox** objektu. Tento objekt vrací hodnoty **true** nebo **false** podle toho, jestli chce uživatel vyhledávat danou barvu. Dále je pro každou barvu

z palety vytvořený objekt **TrackBar**, jehož výčet celočíselných hodnot je v intervalu $< 0, 4 >$, podle kterého je vyhledávána množstevní hodnota výskytu barvy v obrázku.

Vybere-li si uživatel kritéria, podle kterých chce mít obrázky vyhledávány, je nutné pro tyto kritéria vytvořit korektní příkaz do databáze, jehož výsledkem bude výčet všech obrázků z databáze podle zadaných kritérií. Je nezbytné proto vytvořit **event**, jenž bude použtý vždy, když uživatel zvolí kterékoliv kritérium pro vyhledávání. Tento *event* vždy tvoří korektní *select*, který je následně poslán do databáze.

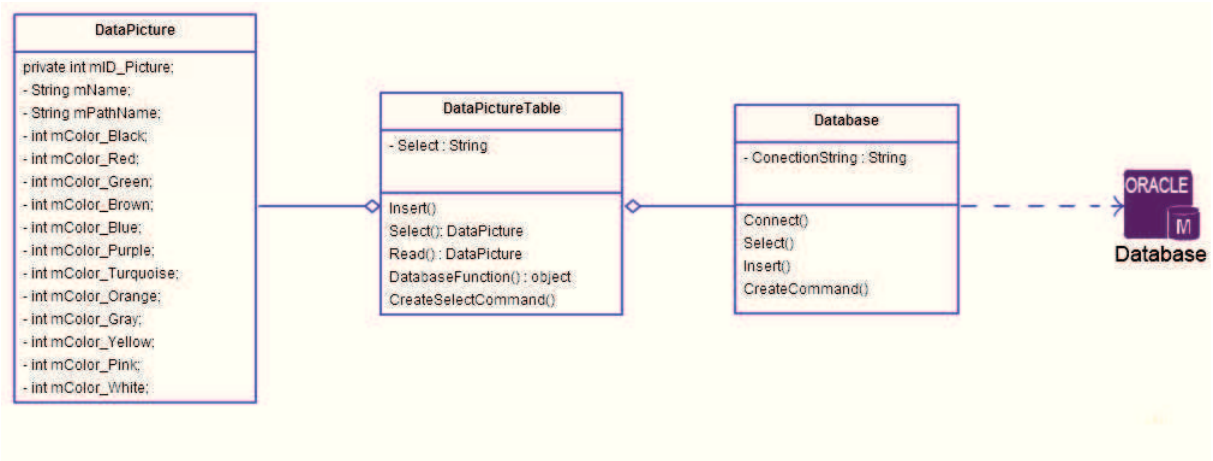
Třída ViewPicture: Zobrazuje uživateli výčet obrázku z databáze dle kritérií zvolených v třídě *SearchPicture*. Třída *SearchPicture* posílá příkaz do databáze a ta následné vrací pole obrázku do třídy *ViewPicture*.

Dané pole obrázku zobrazujeme uživateli buďto všechny najednou, nebo po částech. Záleží na tom, jestli je pole obsáhlejší, tzn. dotaz vrátil mnoho obrázků. Mezi těmito částmi můžeme přepínat jako mezi stránkami. Pro optimální zobrazení uživateli je využívána metoda, zmenšující nám každý obrázek na miniaturu, ale poměr stran obrázku zůstane zachován. Nebylo by efektivní zobrazovat výsledky ve skutečném rozlišení. Pro náhled ve skutečném rozlišení je zde pro každý zobrazený obrázek vytvořený **event** na kliknutí myší na obrázek, který uživateli zobrazí obrázek v plném rozlišení a výčet hodnot všech barev z palety podle zastoupení množství v obrázku.

5.3 Práce s databází

Pro spojení s databází jsem aplikoval návrhový vzor **Data Mapper**[9]. Tento vzor využívá techniky *ORM* (Objektově relační model) zprostředkovávající komunikaci mezi aplikací a databází. Návrhový vzor **Data Mapper** tvoří tyto třídy:

- DataPicture
- TablePicture
- Database



Obrázek 13: Třídní diagram vzoru Data Mapper

Každá z těchto tříd provádí operace, nezbytné pro převod informací z obrázku do databázového uložiště. Třídy jsou navzájem propojené a jedna připravuje informace tak, aby je druhá třída měla neustále k dispozici a mohla s nimi pracovat až do fáze ukládání nebo naopak načítání z databáze.

Třída Database: je hlavní třídou zajišťující fyzické spojení s databází a vyvolává požadované příkazy do databáze.

Třída TablePicture: uchovává přístupové metody pro práci s daty v databázi. Když chceme provést jednu z operací (select, insert, update, delete) je užita metoda této třídy předávající příslušné data třídě Database, která tyto data následně fyzicky pošle do databáze.

Třída DataPicture: uchovává proměnné tak, aby reprezentovaly jednu tabulku z databáze. Rozumíme tím, že uchovává nebo upravuje data pro jejich následné vkládání či načítání z příslušné tabulky v databázi.

6 Experimenty

Závěrečný experiment byl provedený za účelem srovnání mého vyhledávacího systému s podobným systémem od společnosti **google** [8]. Bohužel tento systém neumožňuje vyhledávání několika barev zároveň. Srovnávací výsledky se proto vždy týkají pouze jedné barvy. V následující tabulce můžeme vidět rozdíly ve vyhledávání.

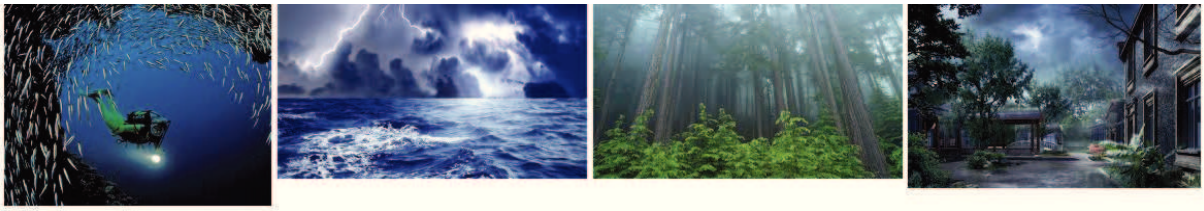
Barva	google	můj systém	rozdíly v %	lidé - nenalezené obrázky
Bílá	50	49	2 %	0 z 1
Černá	50	50	0 %	N
Červená	50	39	22 %	9 z 11
Fialová	50	42	16 %	6 z 8
Hnědá	50	48	4 %	1 z 1
Modrá	50	44	12 %	4 z 6
Oranžová	50	45	10 %	4 z 5
Růžová	50	50	0 %	N
Šedá	50	50	0 %	N
Tyrkysová	50	33	34 %	12 z 17
Zelená	50	50	0 %	N
Žlutá	50	30	40 %	13 z 20
Celkem	600	530	11.66%	49 z 69

Tabulka 5: Rozdíly mezi vyhledávacími systémy

V tabulce můžeme vidět největší rozdíly u barvy **tyrkysové, žluté a červené**. Velké odchylky mohly vzniknout z důvodu klamání lidského oka. Je totiž poměrně složité od sebe odlišit barvu **tyrkysovou** od klasické **modré**. Podobné obtíže ovšem mohou nastat u jakékoliv barvy. Mnohdy jsou odstíny natolik sporné, že zasahují i do čtyř různých barev.

Dalším faktorem rozdílných výsledků může být i hranice minimálního výskytu barvy v obrázku. Můj program považuje hledanou barvu za vyskytující se pouze v případě, že se v obrázku nachází minimálně 10 % barvy. U systému **google** může být výskyt daleko nižší, což nás často přivádí ke zcela rozdílným hodnotám. Z tohoto důvodu jsem se rozhodl nechat zhodnotit právě ty obrázky, ve kterých můj systém hledanou barvu nenašel vůbec. Vybrané obrázky jsem zaslal lidem, kteří se podíleli na hlavním experimentu. Tito lidé podle svého vnímání rozhodli, jestli se hledaná barva v obrázku vyskytuje nebo ne. Výsledné odpovědi ukázaly, že v mnoha případech se hledaná barva v obrázků moc nevyskytovala. Tyto hodnoty lze vidět v posledním sloupci v tabulce 5. Příkladem jsou následující obrázky obsahující barvu s největším procentuálním rozdílem znázorněným v *tabulce 5*.

Obrázek 14: Obrázky s výskytem žluté barvy



Obrázek 15: Obrázky s výskytem tyrkysové barvy

Obrázek 16: Obrázky s výskytem červené barvy

Zde můžete sami posoudit, jestli se hledaná barva v obrázku opravdu vyskytuje či nikoliv.

7 Závěr

Cílem mé bakalářské práce bylo prozkoumat již existující metody a následně naimplementovat svou vlastní metodu nebo jen využít existující. Ve svém vyhledávacím systému jsem tyto dvě varianty zkombinoval.

Jako výchozí barevný model jsem zvolil RGB, jelikož barevné výstupy z většiny monitorů jsou právě v tomto formátu. Tímto nemusela být řešena žádná transformace mezi modely. Zde jsem následně využil existující Voxelovou metodu reprezentující barvu jedním Voxelem a rozšířil jsem ji o výsledky získané v hlavním experimentu. Tímto jsem vlastně vytvořil vylepšenou verzi Voxelové metody díky níž je nyní jedna barva zastupována několika Voxely.

Tento krok se ve finální verzi těší velkým přínosem. V experimentální části, kde jsem porovnával výsledky vyhledávání obrázků mého systému se systémem **google**, jsem získal velmi kladné ohlasy. I ty nejrozdílnější výsledky byly následně lidmi vyhodnoceny ve prospěch mého systému.

V každém případě i můj vyhledávací systém má své zápory, které lze vidět v celkovém počtu odstínů RGB modelu. Není totiž možné přiřadit všechny odstíny k určité barvě v paletě. I za předpokladu, že by se tohle podařilo, vznikaly by chyby ve vyhledávání.

Z toho důvodu bych chtěl vyzvednout model HSL, který je pro vnímání barevných odstínů lidským okem vhodnější. Výsledky tohoto modelu by pro detekci barev v obrázku mohly být přesnější. Tuto metodu jsem však nepoužil, z důvodu složitého převádění z RGB modelu do HSL.

Co se však týče segmentace obrázku, zlatý řez byl nejvhodnějším řešením při obtížích s rozdělováním obrázku na oblasti. Nemuseli zde nastupovat žádné jiné, výpočtově složitější algorytmy, které by obrázek rozdělily na ještě více oblastí.

Pro závěrečné hodnocení bakalářské práce mi poslouží výsledky ze srovnání se systémem **google**. Na základě těchto výsledků mohu konstatovat, že jsem všechny stanovené cíle splnil.

8 Použitá literatura

Reference

[1] software.intel.com *"Color models"*, [ONLINE],
`http://software.intel.com/sites/products/documentation/hpc/ipp/ippi/ippi_ch6/ch6_color_models.html`

[2] nuane.com *"Create color palette"*, [ONLINE],
`http://nuane.com/vyhen/09/palety.html`

[3] web.vscht.cz *"Barevné modely"*, [ONLINE],
`http://web.vscht.cz/kalcicoa/POCPRE/`

[4] brucelindbloom.com *"mathematical conversion of color models"*, [ONLINE],
`http://www.brucelindbloom.com/`

[5] brucelindbloom.com *"Regionální segmentace a shlukování"*, [ONLINE],
`http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/06\%20-\%20Regionalni\%20segmentace\%20a\%20shlukovani.pdf`

[6] fit.vutbr.cz *"Segmentation picture"*, [ONLINE],
`http://www.fit.vutbr.cz/~spanel/segmentace/`

[7] cw.felk.cvut.cz *"Vector distance"*, [ONLINE],
`http://cw.felk.cvut.cz/lib/exe/fetch.php/courses/y336vd/prednasky/p7-cluster.pdf`

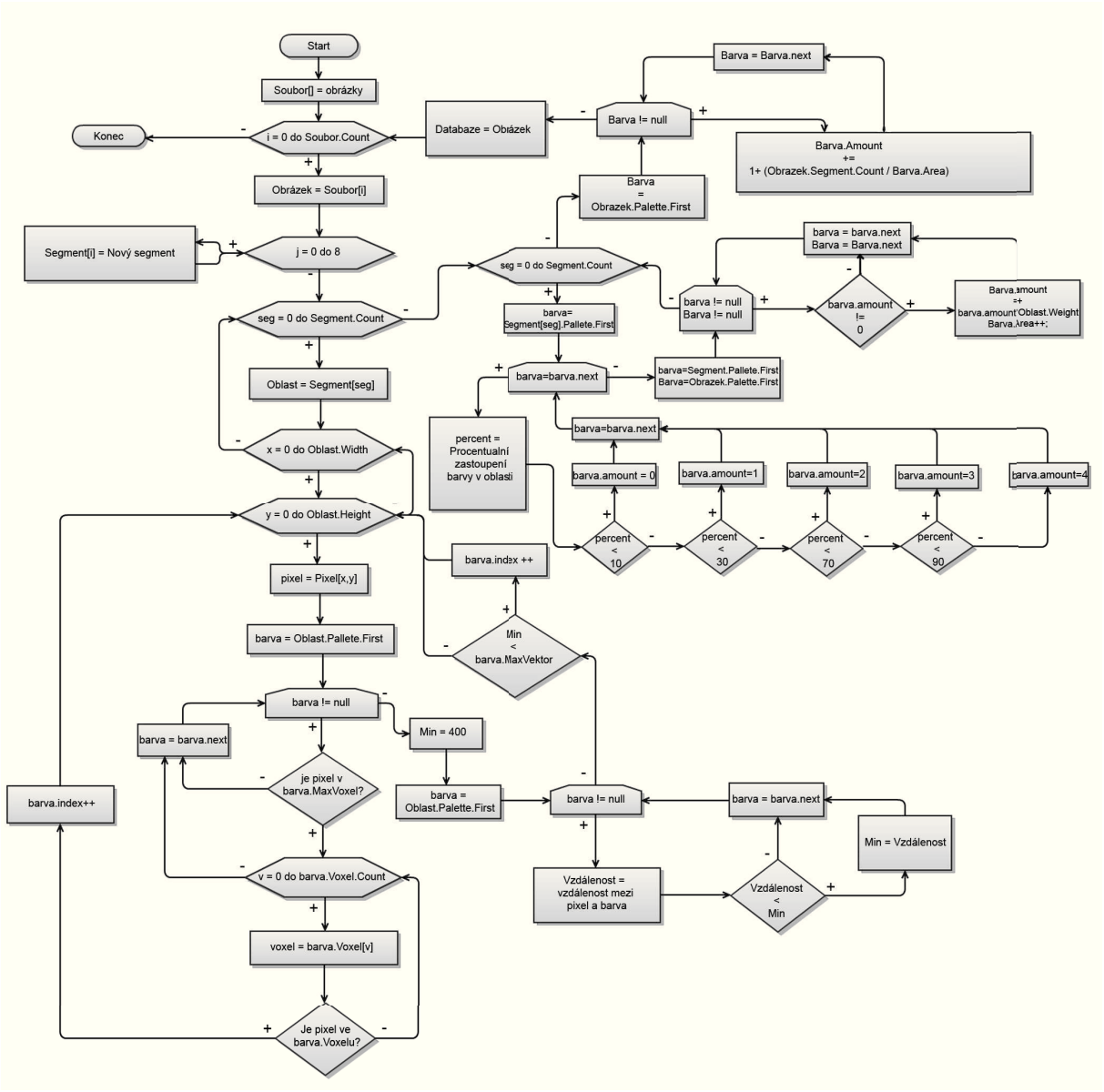
[8] google.com *"Systém na hledání obrázku"*, [ONLINE],
`https://www.google.cz/imghp?hl=cs&tab=wi`

[9] owenou.com *"Data Mapper"*, [ONLINE],
`http://owenou.com/2011/09/24/poeaa-on-rails.html`

[10] N. Krishnan, M. Sheerin Banu, C. Callins Christiyana **Content Based Image Retrieval using Dominant Color Identification Based on Foreground Objects**.
International Conference on Computational Intelligence and Multimedia Applications 2007.

[11] M. Němec **Barvy a barevné systémy**.
Základy počítačové grafiky.

A Vývojový diagram



B Ukázka výsledků detekce barev v obrázcích